



Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño
ccarrenovi@gmail.com



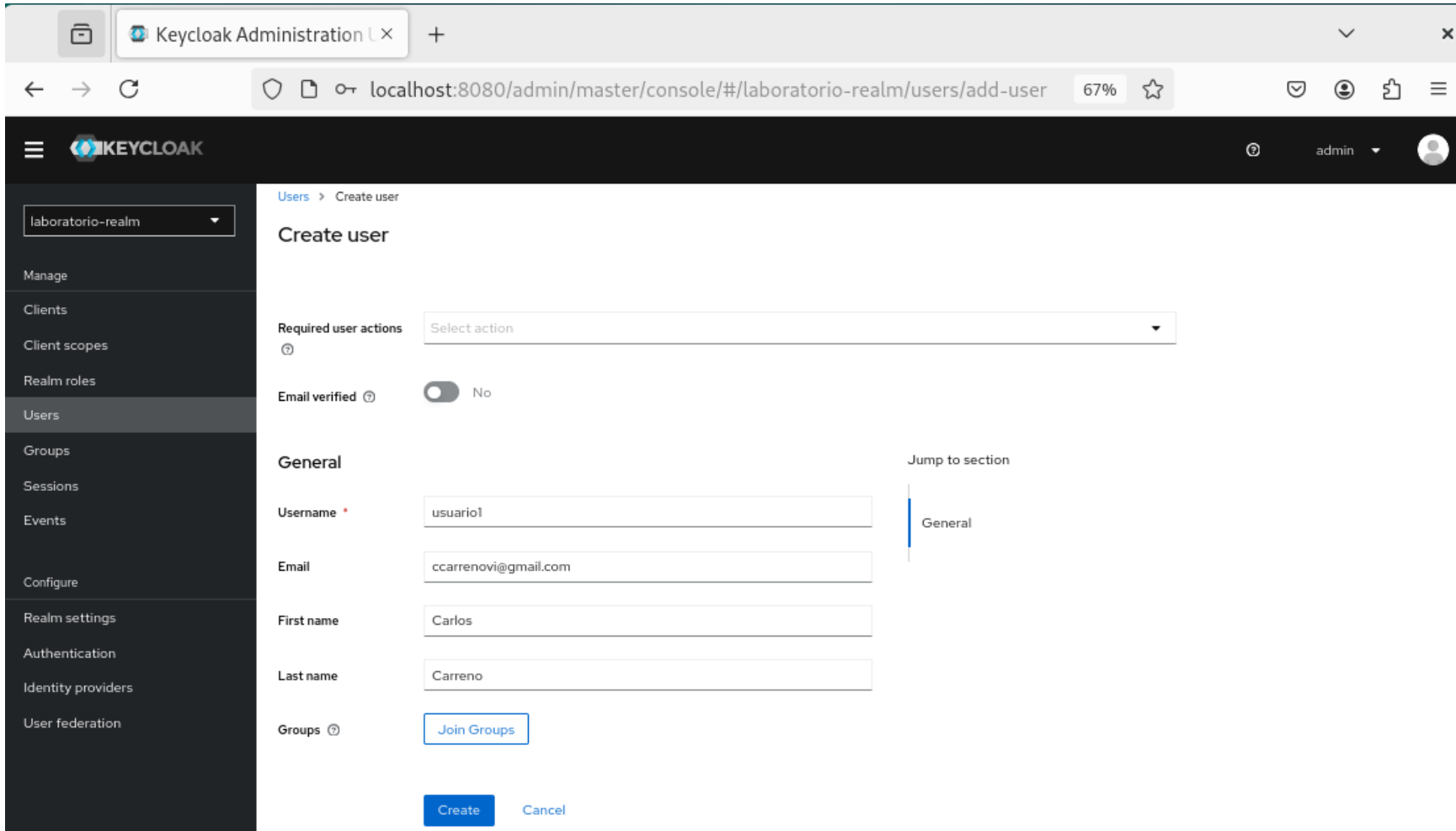
Modulo 7 Gestión de usuarios y autenticación

Objetivo: Introducción a Red Hat SSO y su integración con el desarrollo de aplicaciones con Spring Boot

Duración: 1h



Administración de usuarios en RH-SSO



The screenshot displays the Keycloak Administration console interface. The browser's address bar shows the URL `localhost:8080/admin/master/console/#/laboratorio-realm/users/add-user`. The left sidebar contains a navigation menu with the following items: Manage, Clients, Client scopes, Realm roles, Users (selected), Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'Create user' and includes the following fields and controls:

- Required user actions:** A dropdown menu currently showing 'Select action'.
- Email verified:** A toggle switch set to 'No'.
- General section:**
 - Username:** `usuario1`
 - Email:** `ccarrenovi@gmail.com`
 - First name:** `Carlos`
 - Last name:** `Carreno`
 - Groups:** A button labeled 'Join Groups'.
- Jump to section:** A vertical list with 'General' selected.
- Buttons:** 'Create' and 'Cancel' buttons at the bottom.

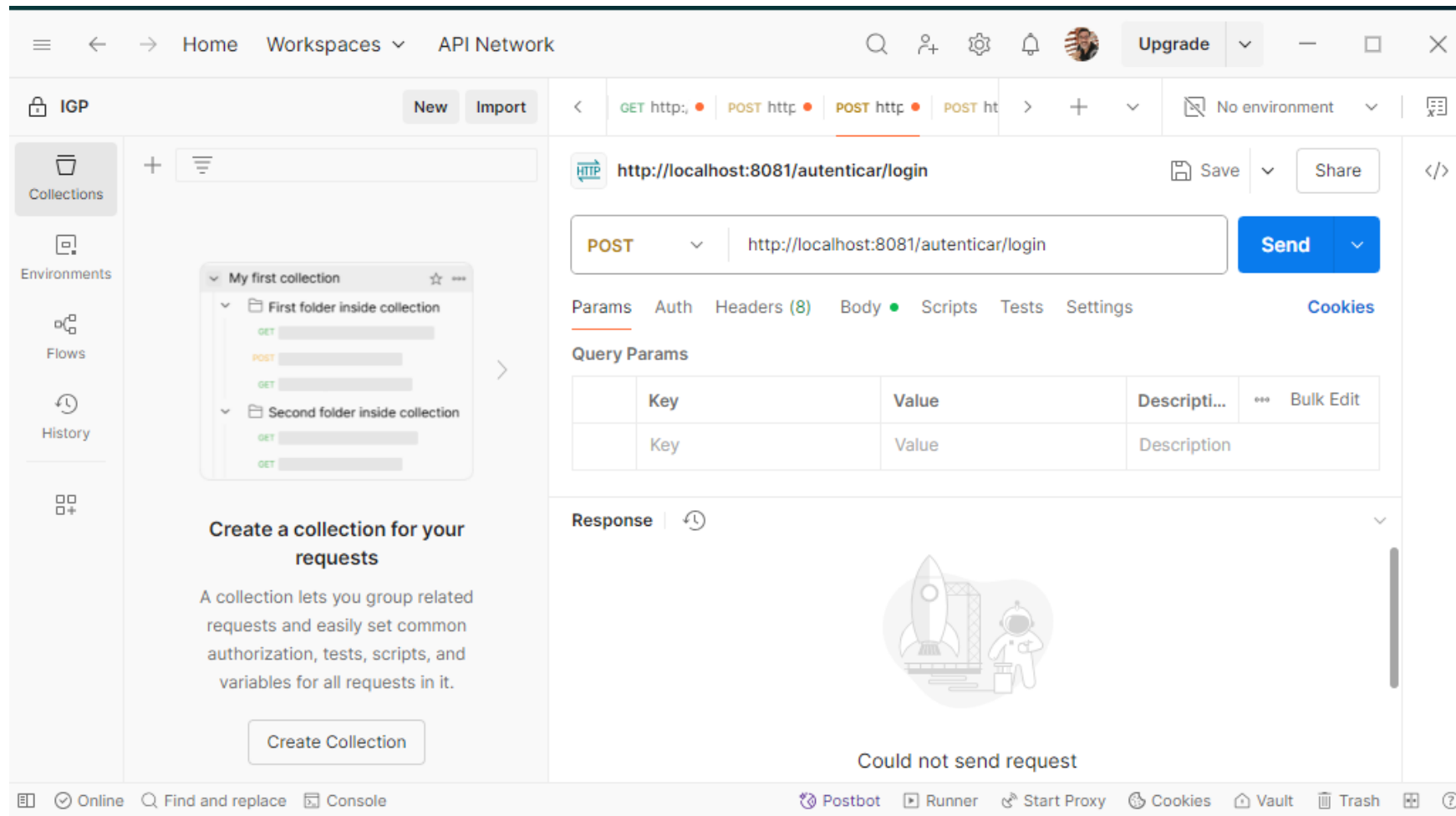


Login interactivo con OAuth2 Authorization Code Flow

1. Redirección al servidor de autorización (Keycloak / RH SSO),
2. Consentimiento del usuario,
3. Intercambio de código por token,
4. Validación del ID Token / Access Token en tu app Spring Boot.



Uso de Postman y CURL para obtener tokens



Validación manual de JWT

Cuando integras Spring Boot con Keycloak o Red Hat SSO, puedes necesitar validar manualmente los tokens JWT, especialmente para escenarios como:

- ✓ Microservicios que no usan Spring Security
- ✓ Validación en servicios de backend sin contexto web
- ✓ Procesamiento de tokens en workers o servicios batch



Clase Utilitaria JwtValidator

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SecureController {

    private final JwtValidator jwtValidator;

    public SecureController(JwtValidator jwtValidator) {
        this.jwtValidator = jwtValidator;
    }

    @GetMapping("/secure-data")
    public String getSecureData(@RequestHeader("Authorization") String authHeader) {
        try {
            String token = authHeader.replace("Bearer ", "");
            JWTClaimsSet claims = jwtValidator.validateToken(token);

            // Acceso a claims personalizados
            String username = claims.getStringClaim("preferred_username");

            return "Datos seguros para: " + username;
        } catch (Exception e) {
            throw new RuntimeException("Token inválido: " + e.getMessage());
        }
    }
}
```



Validación y Spring Security

- Si estás usando Spring Security, puedes acceder al token validado directamente:

```
@GetMapping("/user-info")
public Map<String, Object> getUserInfo(@AuthenticationPrincipal Jwt jwt) {
    return jwt.getClaims();
}
```

- La validación manual es útil cuando no puedes usar el stack completo de Spring Security, pero en la mayoría de los casos es preferible usar los mecanismos integrados.

